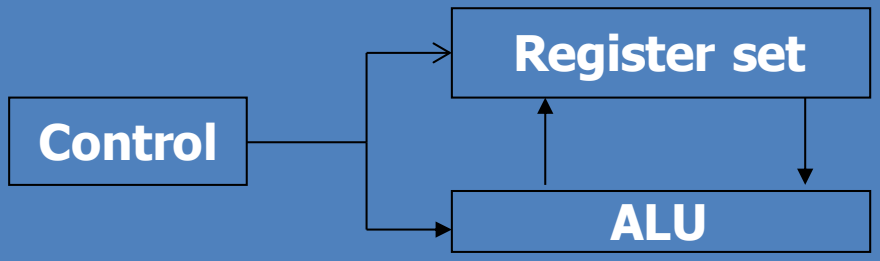
Unit 3
Central Processing Unit

Introduction



- The CPU is made up of 3 major Components.
- The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.
- In programming, memory locations are needed for storing pointers, counters, return addresses, temporary result , etc. Memory access is most time consuming operation in a computer.
- It is then more convenient and more efficient to store these intermediate values in processor registers, which are connected through common bus system.

Accumulator based CPU

Characteristics :

- It is a simple CPU in which the accumulator contains an operand for the Instruction.
- The instruction leaves the result in the accumulator.
- These CPUs have zero address & single address instruction.

Example:

ADD X AC ← AC + M[X]

Accumulator based CPU cont…

The Advantages :

- Short Instruction & less memory space.
- Instruction cycle takes less time because it saves time in Instruction fetching due to the absence of operand fetch.

The Disadvantages :

- Program Size increases, memory size increases.
- Program execution time increases due to increase in program size.

Register based CPU

Characteristics:

- Multiple registers are used.
- The use of registers result in short programs with limited instructions.

Example: **ADD R1, R2, R3** **R1 ← R2 + R3**

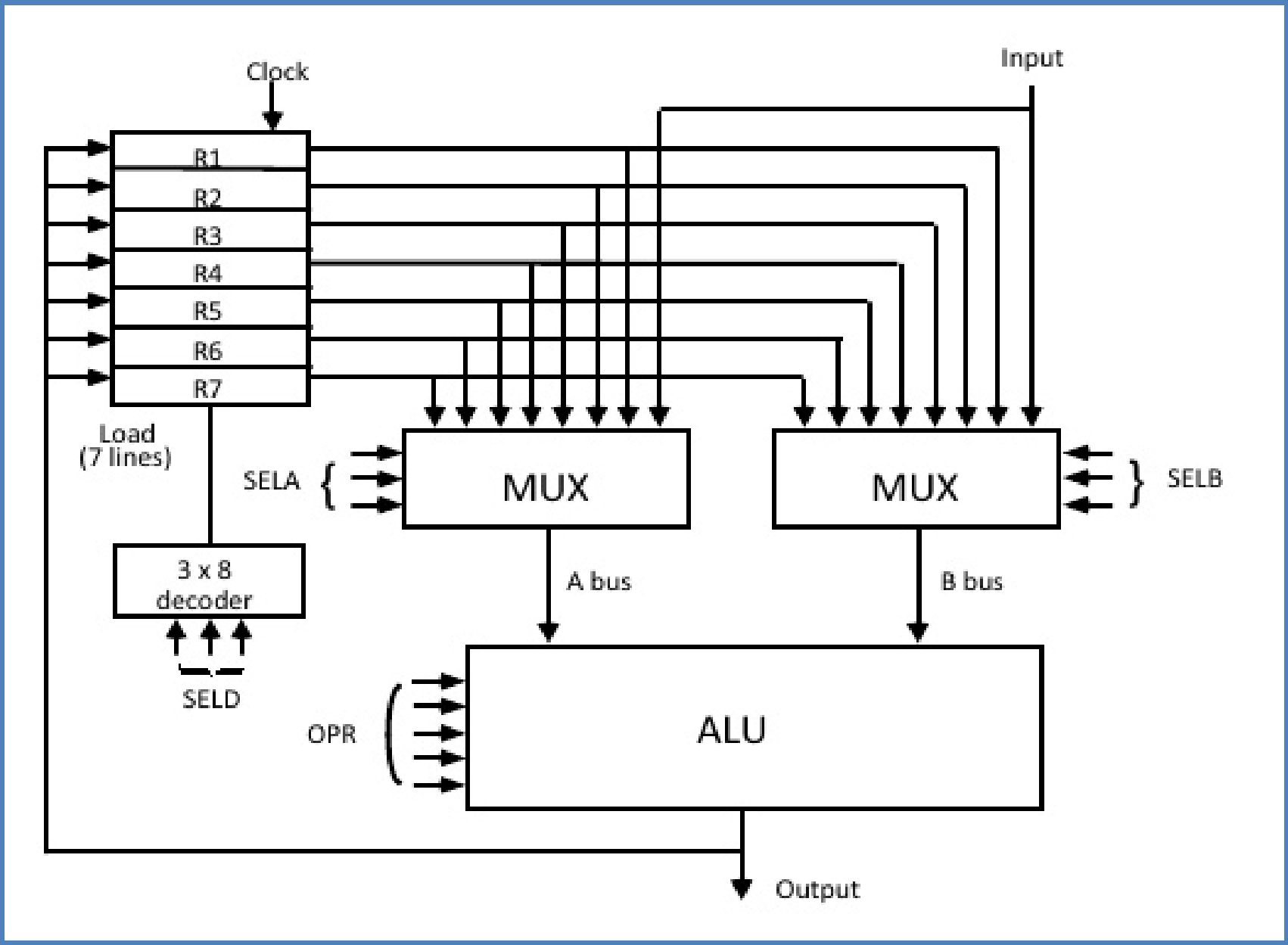
The Advantages :

- Shorter Program size
- Increase in the number of registers, increases CPU efficiency.

The Disadvantages :

- Additional memory accesses are needed.
- Compilers need to be more efficient in this aspect

General Register Organization



Example of Micro-operations

R1 ←R2+R3

To Perform this operation, the **control must provide binary selection variables** to the following selector inputs:

1. MUX A selector **(SELA)**: to place the content of R2 into bus A.
2. MUX B selector **(SELB)**: to place the content of R3 into bus B.
3. ALU operation selector **(OPR)**: to provide the arithmetic addition A + B.
4. Decoder destination selector **(SELD)**: to transfer the content of the output bus into R1.

Example of Micro-operations Cont…

Control Word

There are therefore **14 binary selection inputs in the unit** , and their combined value specifies a *control word*

Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

- When **SELA** or **SELB** is **000** the corresponding multiplexer select the external input data.
- When **SELD** is **000**, no destination register is selected but the content of the output bus are available in the external output.

Encoding of ALU Operation

OPR Select	Operation	SELD
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A – B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Example of the Micro-operations for the CPU

Symbolic Designation					
Micro-operations	SELA	SELB	SELD	OPR	Control Word
R1← R2 - R3	R2	R3	R1	SUB	010-011-001-00101
R4← R4 V R5	R4	R5	R4	OR	100-101-100-01010
R6← R6 + 1	R6	-	R6	INCRA	110-000-110-00001
R7← R1	R1	-	R7	TSFA	001-000-111-00000
Output← R2	R2	-	None	TSFA	010-000-000-00000
Output ← Input	Input	-	None	TSFA	000-000-000-00000
R4← shl R4	R4	-	R4	SHLA	100-000-100-11000
R5← 0	R5	R5	R5	XOR	101-101-101-01100

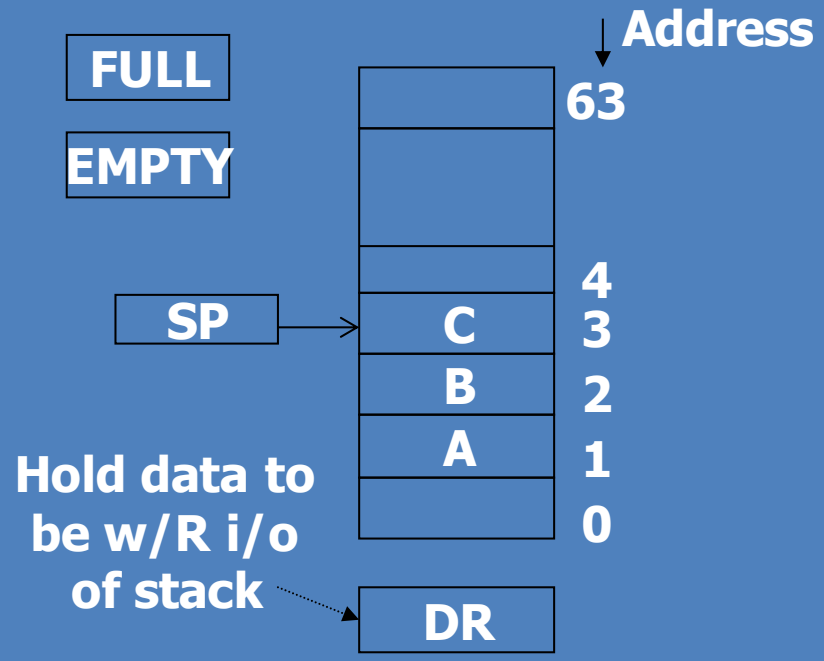
Stack Organization

- The **stack** in digital computers is essentially a memory unit with an **address register** that count only after an initial value is loaded into the stack.
- The Register that hold the address for the stack is called **stack pointer(SP).**
- SP value always point at the top item in the stack.
- The 2 operations of stacks are the **insertion (push)**, and **deletion (pop)** of items.
- A stack can be organized as a collection of a finite number of memory words or registers.

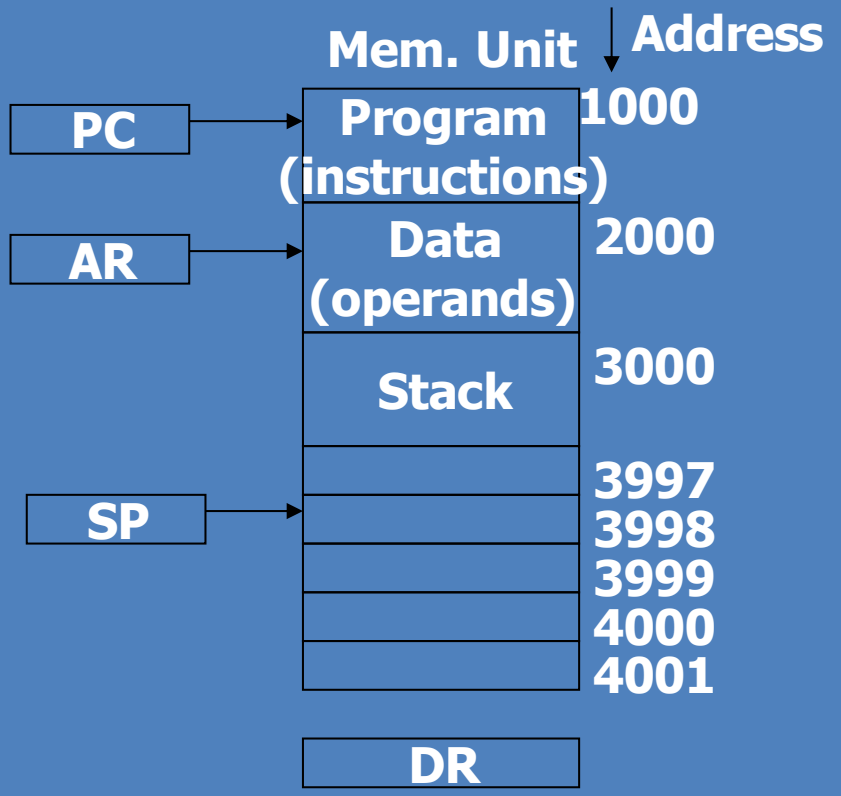
Stack Organization

- In a 64-word stack, SP contains 6 bits because $2^6 = 64$.
- SP has only six bits, it cannot exceed a number grater than 63(111111 in binary).
- When 63 is incremented by 1, the result is 0 since 111111 + 1 =1000000 in binary, but SP can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 111111.
- The 1-bit register FULL is set to 1 when stack is full.
- The 1-bit register EMTY is set to 1 when stack is empty.
- DR is data register that holds the binary data to be written into or read out of the stack.

Stack Organization



Block diagram of
a 64 word-register stack
6 bit SP



Computer memory with
program, data & stack segments

Stack Organization(Operations)

Push operation

- Initially, SP is cleared to 0, Emty is set to 1, and Full is cleared to 0,
- SP points to the word at address 0 (Zero) and the stack is marked empty and not full. if the stack is not full , a new item is inserted with a push operation.

STEP1: SP ← SP-1 Decrement stack pointer

STEP2: M[SP] ← DR Write item on top of the stack

STEP3: If (SP=0) then (FULL ← 1) Check if stack is full

(when 63 is incremented by 1, the result is 0.)

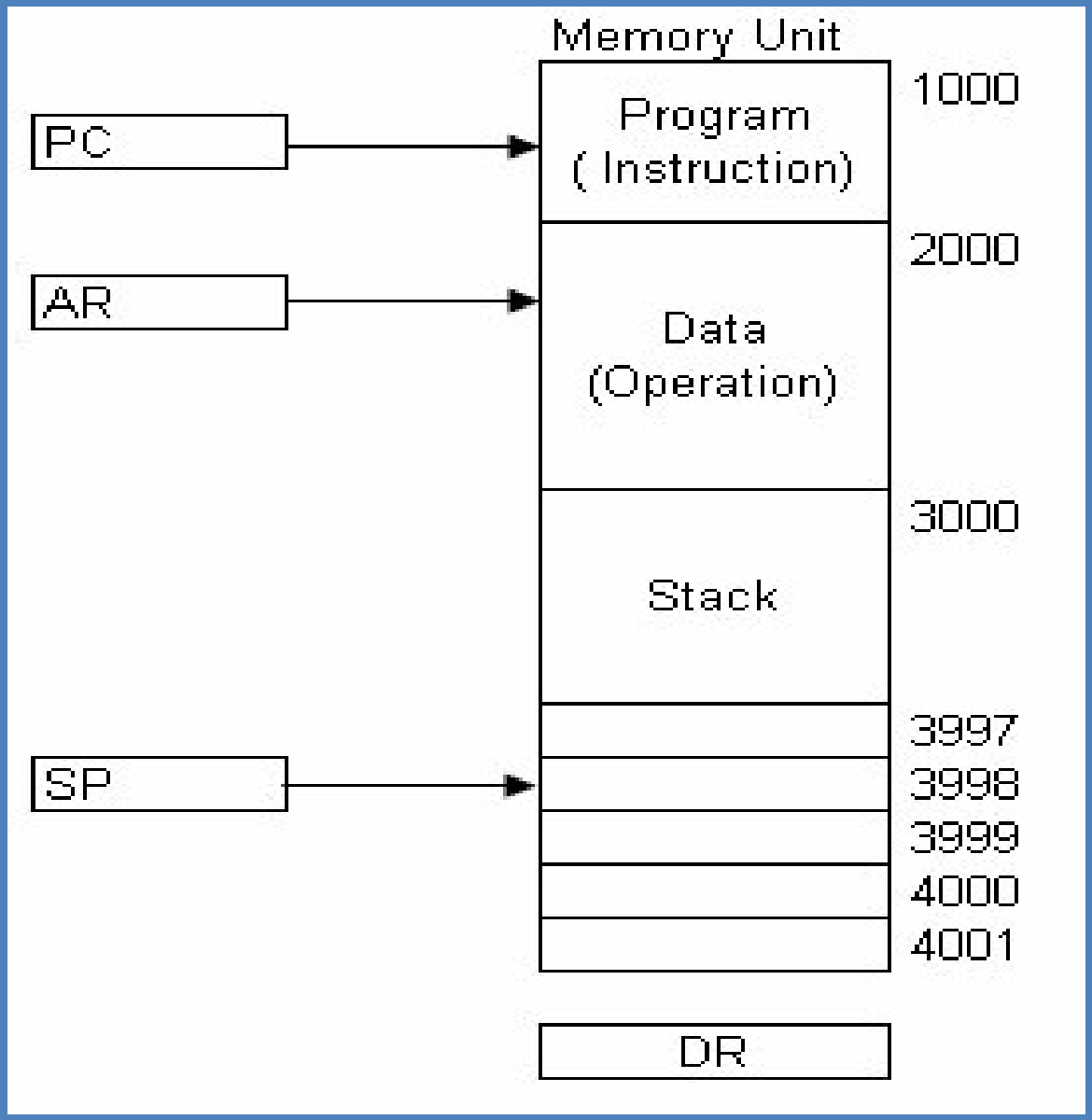
STEP4: EMTY ← 0 Mark the stack not empty

Accumulator based CPU

The pop operation consists the following sequence of micro-operations:

STEP:1	DR ← M[SP]	Read item from top of the stack
STEP2:	SP ← SP+1	Increment stack pointer
STEP3:	If (SP=0) then (EMPTY ← 1)	Check if stack is empty
STEP4:	FULL ← 0	Mark the stack not full

Stack Organization (In Memory stack)



Stack Organization (In Memory stack) Cont…

- A stack can be implemented in a portion of a random –access memory attached to a CPU, and using a processor register as SP as shown the diagram above.
- In the diagram shown Computer memory with program, data & stack segments:
  - PC (points at the address of the next instruction) is used during the fetch phase to read an instruction.
  - AR (points at an array of data) is used during the execution phase to read an operand.
  - SP is used push or pop items into or from the stack.

Stack Organization (In Memory stack) Cont…

- The three registers are connected to a common address bus, & either one can provide an address for memory.
- **Push:** insert (write) new item at the top of the stack

  $SP \leftarrow SP - 1,$

  $M[SP] \leftarrow DR$
- **Pop:** delete (read) the top item from the stack

  $DR \leftarrow M[SP],$

  $SP \leftarrow SP + 1$

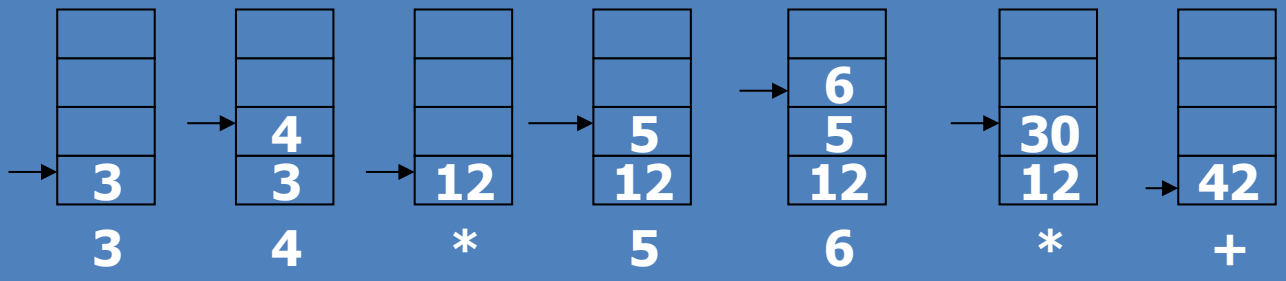
Stack Organization (In Memory stack) Cont…

- The stack limits (underflow/overflow) can be checked by using 2 processor registers:
  - One to hold the upper limit (3000 in this example)
  - Another to hold the lower limit (4001 in this example)
- Two micro-operations are needed for push or pop :
  1. An access to memory through SP
  2. Updating SP

Stack Organization (Reverse Polish Notation)

- A stack organization is very effective for evaluating arithmetic expressions.
- The reverse Polish notation is in a form suitable for stack manipulation, where the infix expression A*B+C*D can be written in Reverse Polish Notation (RPN or postfix notation) as: AB*CD*+
- Example: Stack operation to evaluate 3*4+5*6

  => 34*56*+ (in in reverse Polish notation)



Instruction Formats

The most common fields founded in the instruction formats are:

1. An operation code field that specified the operation to be performed (ADD, SUB, SHIFT, Complement).
2. An address field that designates a memory address or a processor registers.
3. A mode field that specifies the way the operand or the effective address is determined (Addressing modes, Sec. 8.5).

Instruction Formats

Most computers fall into one of three types of CPU organization.

(1) Single Accumulator organization (ADD X; AC <- AC + M [×])

(2) General Register Organization (ADD R1, R2, R3)

(3) Stack Organization (PUSH X)

Instruction Formats

Three-Address Instruction: Computer with three addresses instruction format can use each address field to specify either processor register or a memory operand.

Evaluate X = (A + B) * (C + D)

ADD R1, A, B	R1 ← M [A] + M [B]
ADD R2, C, D	R2 ← M [C] + M [D]
MUL X, R1, R2	M [X] ← R1 * R2

- The advantage of the three address formats is that it results in short program when evaluating arithmetic expression.
- The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Instruction Formats

Two-Address Instruction: Each address field can specify either a processes register on a memory word.

Evaluate X = (A + B) * (C + D)

MOV	R1, A	R1 ← M [A]
ADD	R1, B	R1 ← R1 + M [B]
MOV	R2, C	R2 ← M [C]
ADD	R2, D	R2 ← R2 + M [D]
MUL	R1, R2	R1← R1 * R2
MOV	X, R1	M [X] ← R1

X = (A + B) * (C + D)

Instruction Formats

One-Address Instruction: It used an implied accumulator (AC) register for all data manipulation.

Evaluate X = (A + B) * (C + D)

```
LOAD   A        AC ←M [A]
ADD    B        AC ← AC + M [B]
STORE  T        M [T] ← AC

LOAD   C        AC ← M [C]
ADD    D        AC ← AC + M [D]
MUL    T        AC ← AC * M [T]
STORE  X        M [X] ← AC
```

Instruction Formats

Zero-Address Instruction:

Evaluate X = (A+B) * (C+D)

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP	X	M [X] ← TOS

Addressing Modes

Instruction Cycle divided into three major phases:

1. Fetch the instruction from memory.

   Program Counter (PC)

2. Decode the instruction.

   Determine the Operation, Addressing mode, location of the operand.

3. Execute the instruction.

Addressing Modes

What is Mode Field ?

- The mode field is used to locate the operand needed for the operation.
- There may or may not be an address field in the instruction.
- If there is an address field, it may designate a memory address or a processor register.

Addressing Modes

- Microprocessor executes the instructions stored in memory (RAM).
- It executes one instruction at a time.
- Each of the instruction contains operations and operands.
- Operation specifies the type of action to be performed.
    - For example: ADD, SUB, MOV, INC, LOAD, STORE
- Operands are the data on which the operation is to be performed.
    - MOV B, A Here MOV is operation and (B & A) are operands.
    - ADD B Here ADD is operation and (B) is operand.

Addressing Modes

- Operand can be place either in one of the processor register or in memory.
- There are different ways to get the operands.
- The way in which the operand is taken from register or memory is named as addressing mode.

Addressing Modes

What is Effective Address(EA)?

- In a system without virtual memory, the effective address will be either a main memory address or a register.
- In a virtual memory system, the effective address is a virtual address or a register.

(Note: The actual mapping to a physical address is a function of the memory management unit (MMU) and is invisible to the programmer.)

Addressing Modes

The most well known addressing mode then are:

1. Implied mode.
2. Immediate mode
3. Register mode
4. Register Indirect mode
5. Auto-increment or Auto-decrement mode
6. Direct Mode
7. Indirect Mode
8. Relative Address Mode
9. Index Addressing Mode
10. Base Register Addressing Mode.

Implied Modes

In this mode operands are specified implicitly in the definition of the instruction.

- The stack mode of addressing is a form of implied addressing.
    - The machine instructions need not include a memory reference but implicitly operate on the top of the stack.

Implied Modes

Examples:

1.Complement Accumulator (CMA)

(Here accumulator A is implied by the instruction)

2. Complement Carry Flag (CMC)

(Here Flags register is implied by the instruction)

3. Set Carry Flag (STC)

(Here Flags register is implied by the instruction)

Immediate Modes

- The simplest form of addressing is immediate addressing, in which the operand value is present in the instruction.
- This mode can be used to define and use constants or set initial values of variables.
- The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle.
- The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length.

Immediate Modes

- The operand is specified with in the instruction.
- Operand itself is provided in the instruction rather than its address.

Example

Move Immediate

MVI A , 15h A ← 15h Here 15h is the immediate operand

Add Immediate

ADI 3Eh A ← A + 3Eh Here 3Eh is the immediate operand

Register Modes

- In this mode the operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction (2^K).

Example : MOV A, R4

- The advantages of register addressing are that
  (1) only a small address field is needed in the instruction, and
  (2) no time-consuming memory references are required.
- The disadvantage of register addressing is that the address space is very limited.

Register Modes

Example:

Move

MOV C , A C ← A Here A is the operand specified in register

Add

ADD B A ← A + B Here B is the operand specified in register

Register Indirect Modes

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- In other words, selected register contains the address of the operand rather than the operand itself.
- The advantages of this mode is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Register Indirect Modes

Example:

Move

MOV A , M A ← [[H][L]] It moves the data from memory location specified by HL register pair to A.

A	
H	28
L	05

2807	
2806	
2805	A9
2804	
2803	
2802	
2801	
2800	

A	A9
H	28
L	05

2807	
2806	
2805	A9
2804	
2803	
2802	
2801	
2800	

Direct Addressing Mode

- The instruction specifies the direct address of the operand.
- The memory address is specified where the actual operand is.

Example

Load Accumulator

LDA 2805h A ← [2805] It loads the data from memory location 2805 to A.

Store Accumulator

STA 2803h [2803] ← A It stores the data from A to memory location 2803.

Direct Addressing Mode

LDA 2805h A ← [2805]

A

2807	
2806	
2805	5C
2804	
2803	
2802	
2801	
2800	

A	5C

2807	
2806	
2805	5C
2804	
2803	
2802	
2801	
2800	

Direct Addressing Mode

STA 2803h [2803] ← A

A	5C

2807	
2806	
2805	
2804	
2803	
2802	
2801	
2800	

A	5C

2807	
2806	
2805	
2804	
2803	5C
2802	
2801	
2800	

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range.

Indirect Addressing Modes

- The instruction specifies the indirect address where the effective address of the operand is placed.
- The memory address is specified where the actual address of operand is placed.

Example

Move

MOV A, 2802h A ← [[2802]] It moves the data from memory location specified by the location 2802 to A.

Indirect Addressing Modes

MOV A, 2802h A ← [[2802]]

A

2807	
2806	FF
2805	
2804	
2803	06
2802	28
2801	
2800	

A	28

2807	
2806	FF
2805	
2804	
2803	06
2802	28
2801	
2800	

Relative Addressing Mode

- In relative addressing mode, contents of Program Counter PC is added to address part of instruction to obtain effective address.
- The address part of the instruction is called as offset and it can +ve or –ve.
- When the offset is added to the PC the resultant number is the memory location where the operand will be placed.

Relative Addressing Mode

Offset = 04h

PC	2801

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

Effective address of operand = PC + 01 + offset

Effective address of operand = 2801 + 01 + 04

Effective address of operand = 2806h

Index Addressing Mode

- In index addressing mode, contents of Index register is added to address part of instruction to obtain effective address.
- The address part of instruction holds the beginning/base address and is called as base.
- The index register hold the index value, which is +ve.
- Base remains same, the index changes.
- When the base is added to the index register the resultant number is the memory location where the operand will be placed.

Index Addressing Mode

Base = 2800h

Effective address of operand = Base + IX

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0000

2800h + 0000h = 2800h

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0001

2800h + 0001h = 2801h

2807	22
2806	FF
2805	6D
2804	59
2803	08
2802	2E
2801	F3
2800	9F

IX 0002

2800h + 0002h = 2802h

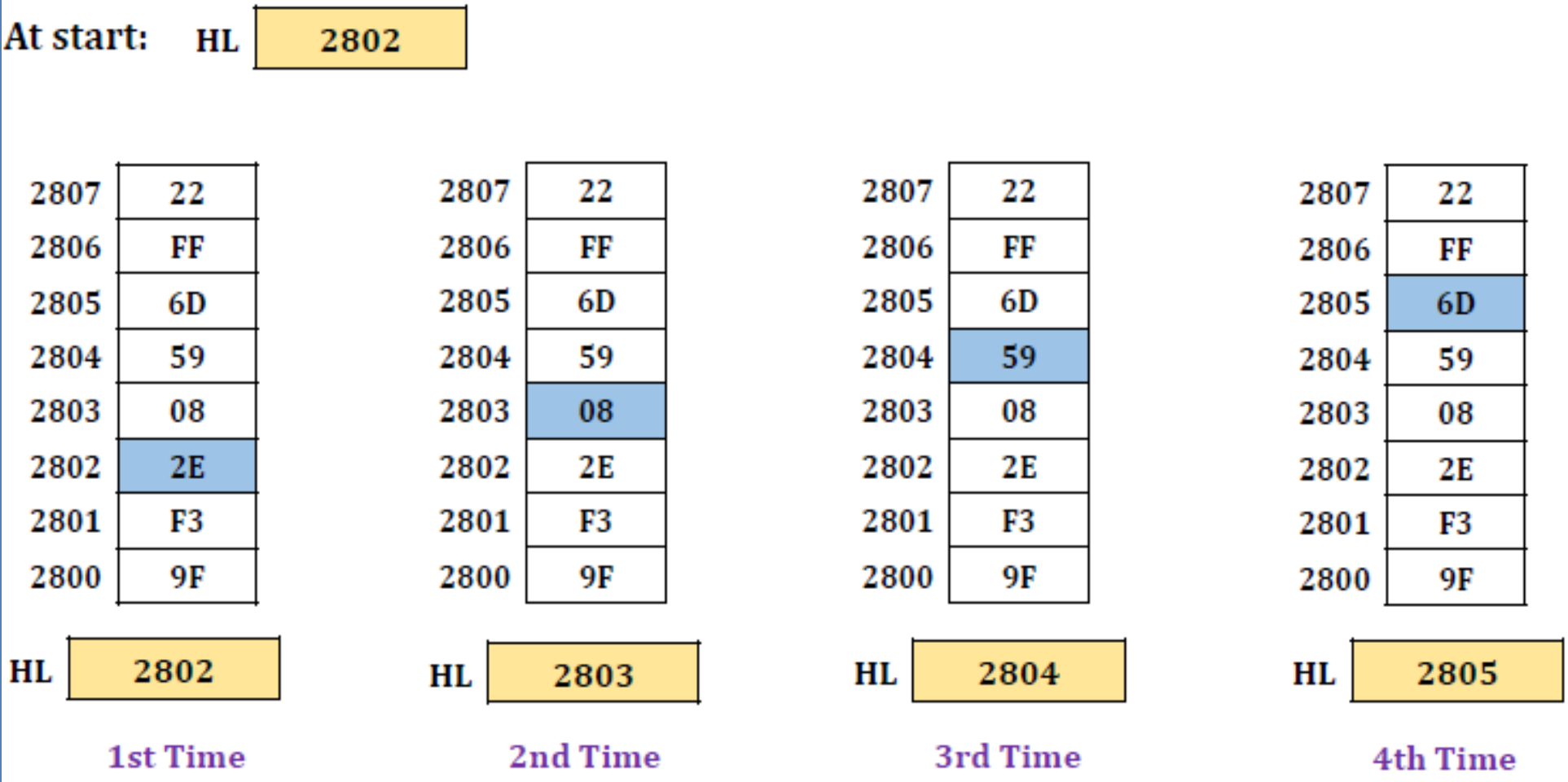
Auto-increment or Auto-decrement Addressing Mode

- It is similar to register indirect addressing mode.
- Here the register is incremented or decremented before or after its value is used.

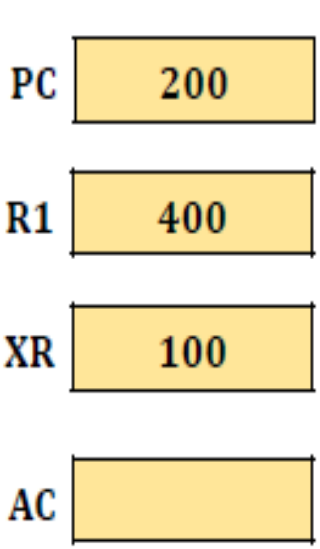
Auto-increment or Auto-decrement Addressing Mode

HL pair incremented after its value is used

At start: HL 2802



Example problem



Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

PC = Program Counter
R1 = Register
XR = Index Register
AC = Accumulator

- Memory is having first instruction to load AC
- Mode will specify the addressing mode to get operand.
- Address field of instruction is 500.

Find out the effective address of operand and operand value by considering different addressing modes.

Example problem

PC	200
R1	400
XR	100
AC	

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

1. Immediate Addressing Mode

- As instruction contains immediate number 500.
- It is stored as address 201.

Effective Address = 201

Operand = 500

AC	500

Example problem

PC	200
R1	400
XR	100
AC	

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

2. Register Addressing Mode

- Register R1 contains 400.
- As operand is in register so no any memory location.

Effective Address = Nil

Operand = 400

AC	400

Example problem

PC	200
R1	400
XR	100
AC	

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

3. Register Indirect Addressing Mode

- Register R1 contains 400.
- So effective address of operand is 400.
- The data stored at 400 is 700.

Effective Address = 400

Operand = 700

AC	700

Example problem

PC	200
R1	400
XR	100
AC	

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

4. Direct Addressing Mode

- Instruction contains the address 500.
- So effective address of operand is 500.
- The data stored at 500 is 800.

Effective Address = 500

Operand = 800

AC	800

Example problem

PC	200
R1	400
XR	100
AC	

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

5. Indirect Addressing Mode

- Instruction contains the address 500.
- Address at 500 is 800.
- So effective address of operand is 800.
- The data stored at 800 is 300.

Effective Address = 800
Operand = 300

AC	300

Example problem

PC	200
R1	400
XR	100
AC	

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

6. Relative Addressing Mode

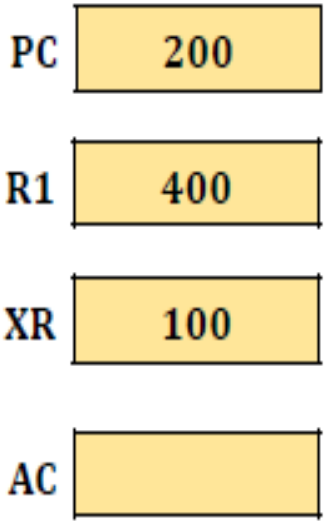
- PC = 200.
- Offset = 500.
- Instruction is of 2 bytes.
- So effective address = PC + 2 + offset = 200 + 500 +2 = 702 .
- The data stored at 702 is 325.

Effective Address = 702

Operand = 325

AC	325

Example problem



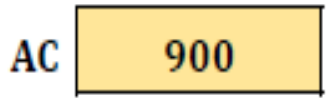
Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

7. Index Addressing Mode

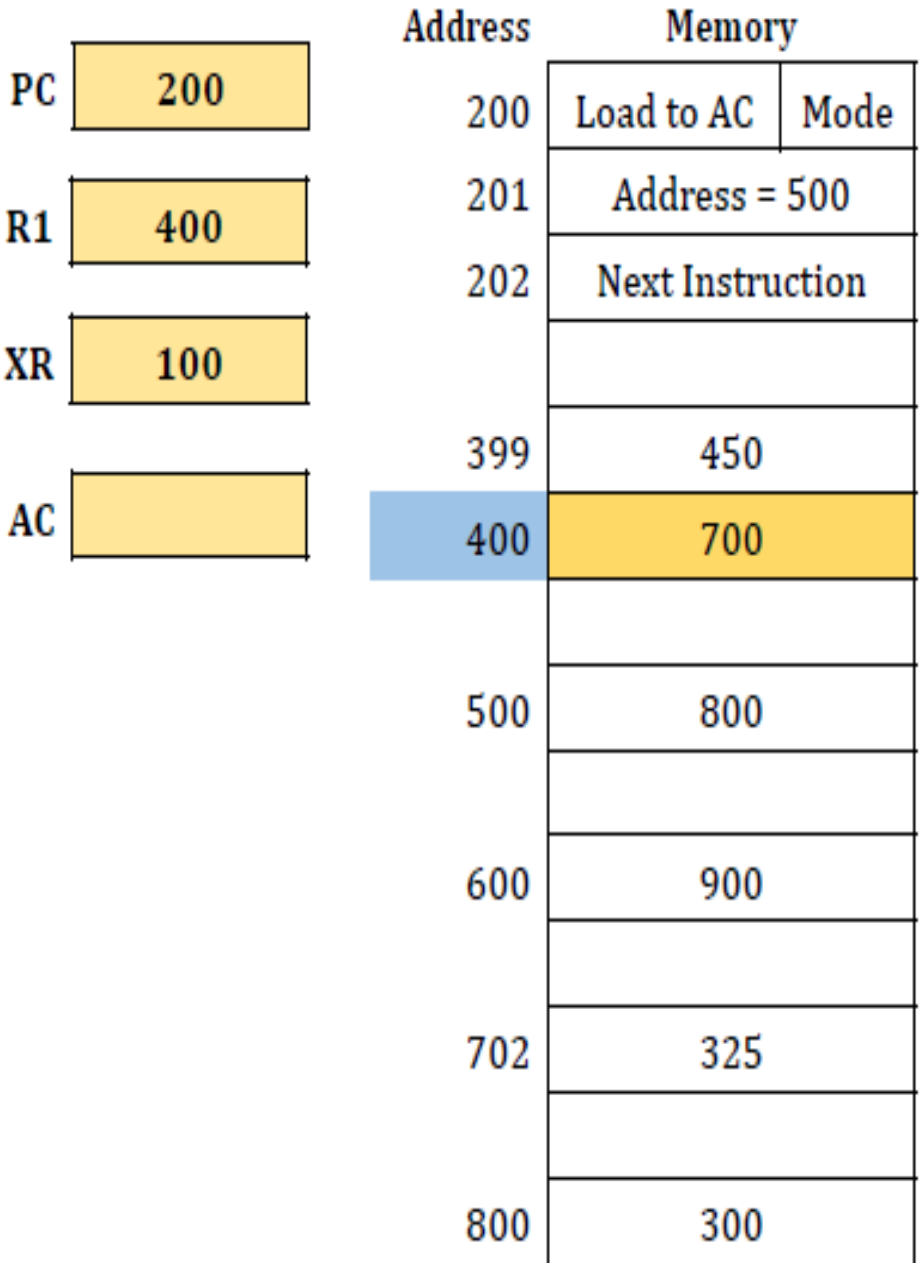
- XR = 100.
- Base = 500.
- So effective address = Base + XR = 500 + 100 = 600 .
- The data stored at 600 is 900.

Effective Address = 600

Operand = 900



Example problem

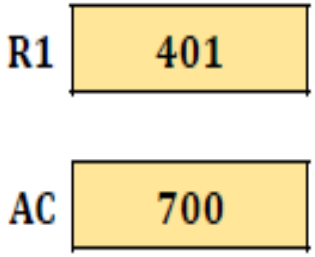


8. Autoincrement Addressing Mode

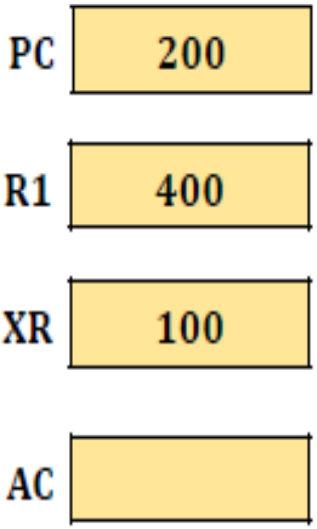
- It is same as register indirect addressing mode except the contents of R1 are incremented after the execution.
- R1 contains 400.
- So effective address of operand is 400.
- The data stored at 400 is 700.

Effective Address = 400

Operand = 700



Example problem



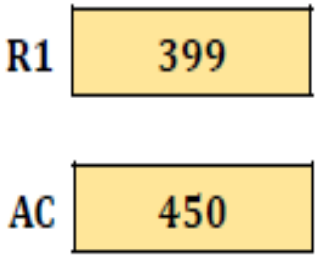
Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

9. Autodecrement Addressing Mode

- It is same as register indirect addressing mode except the contents of R1 are decremented before the execution.
- R1 contains 400.
- R1 is first decremented to 399.
- So effective address of operand is 399.
- The data stored at 399 is 450.

Effective Address = 399

Operand = 450



Data Transfer and Manipulation

Most computer instructions can be classified into three categories:

1) Data transfer,

2) Data manipulation,

3) Program control instructions

Data Transfer Instruction

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.

Data Transfer Instruction

Typical Data Transfer Instructions

NAME	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Transfer Instruction

Typical Data Transfer Instruction :

- **Load :** transfer from memory to a processor register, usually an AC (memory read)
- **Store :** transfer from a processor register into memory (memory write)
- **Move :** transfer from one register to another register
- **Exchange :** swap information between two registers or a register and a memory word
- **Input/Output :** transfer data among processor registers and input/output device
- **Push/Pop :** transfer data between processor registers and a memory stack

Data Transfer Instruction

Data Transfer Instructions with Different Addressing Modes

MODE	ASSEMBLY CONVENTION	REGISTER TRANSFER
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ←R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

Data Manipulation Instructions

- Data Manipulation Instructions perform operations on data and provide the computational capabilities for the computer.

It is divided into three basic types:

1) Arithmetic
2) Logical and bit manipulation,
3) Shift Instruction

Data Manipulation Instructions

Arithmetic Instructions

- The four basic arithmetic operations are **addition, subtraction, multiplication & division**.
- Some computers have only addition & subtraction instruction .The multiplication & division must then be generated by means of **Software Subroutines** (Self contained sequence of instruction that perform a computational task).

Data Manipulation Instructions

A list of Typical arithmetic instruction are as follows:-

NAME	MNEMONICS
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate(2's Complement)	NEG

Data Manipulation Instructions

Arithmetic Instruction:

1. **Increment (INC):-**this instruction adds 1 to the value stored in a register or memory word.
   - One common characteristic of increment operations when executed in processor registers is that a **binary number of all 1's when incremented produces a result of all 0's**.
2. **Decrement (DEC):-**this instruction subtracts 1 from the value stored in a register or memory word.
   - One common characteristic of decrement operations when executed in processor registers is that **a binary number of all 0's when decremented, produces a result of all 1's**.

Data Manipulation Instructions

3. Addition (ADD)
4. Subtract (SUB)
5. Multiply (MUL)
6. Divide (DIV)

- These instructions may be available for different types of data .it may be binary, decimal, floating-point data.
- The mnemonics for three add instructions that specify different data types are shown below:

  ADDI:- Add two binary integer numbers.

  ADDF:- Add two floating-point numbers.

  ADDD:- Add two decimal numbers in BCD.

Data Manipulation Instructions

7. **Add with Carry (ADDC):-** A special carry flip- flop is used to store the carry from an operation. The instruction "add with carry" performs the addition on two operands plus the value of the carry from the previous computational.

8. **Subtract with borrow(SUBB):-** subtracts two words and a borrow which may have resulted from a previous subtract operation.

9. **Negate (2's complement):-**the negate instruction forms the 2's complement of a number effectively.

Data Manipulation Instructions

Logical and Manipulation instructions

- They are useful for **manipulating individual bits or a group of bits** that represent binary-coded information. The logical instruction consider each bit of the operand separately and treat it as a Boolean variable.
- Logical and bit manipulation instructions are as follows:

1. **Clear(CLR):-** the clear instruction causes the specified operand to be replaced by 0's.

2. **Complement (COM):-**the complement instruction produces the 1's complement by inverting all the bits of the operands.

Data Manipulation Instructions

3.AND(AND)

4.OR(OR)

5.Exclusive-OR(XOR)

- The AND,OR,XOR instructions produces the corresponding logical operations on individual bits of the operands.

6. Clear Carry (CLRC):-

7. Set Carry (SETC):-

8. Complement Carry (COMC):

- Individual bits such as a carry can be cleared, set, or complement with appropriate instructions.

Data Manipulation Instructions

9. Enable interrupts (EI):-
10. Disable interrupts (DI):-
    - Flips flops that controls the interrupts facility and is either enabled or disabled by means of bit manipulation instructions.

Data Manipulation Instructions

Shift instructions:-

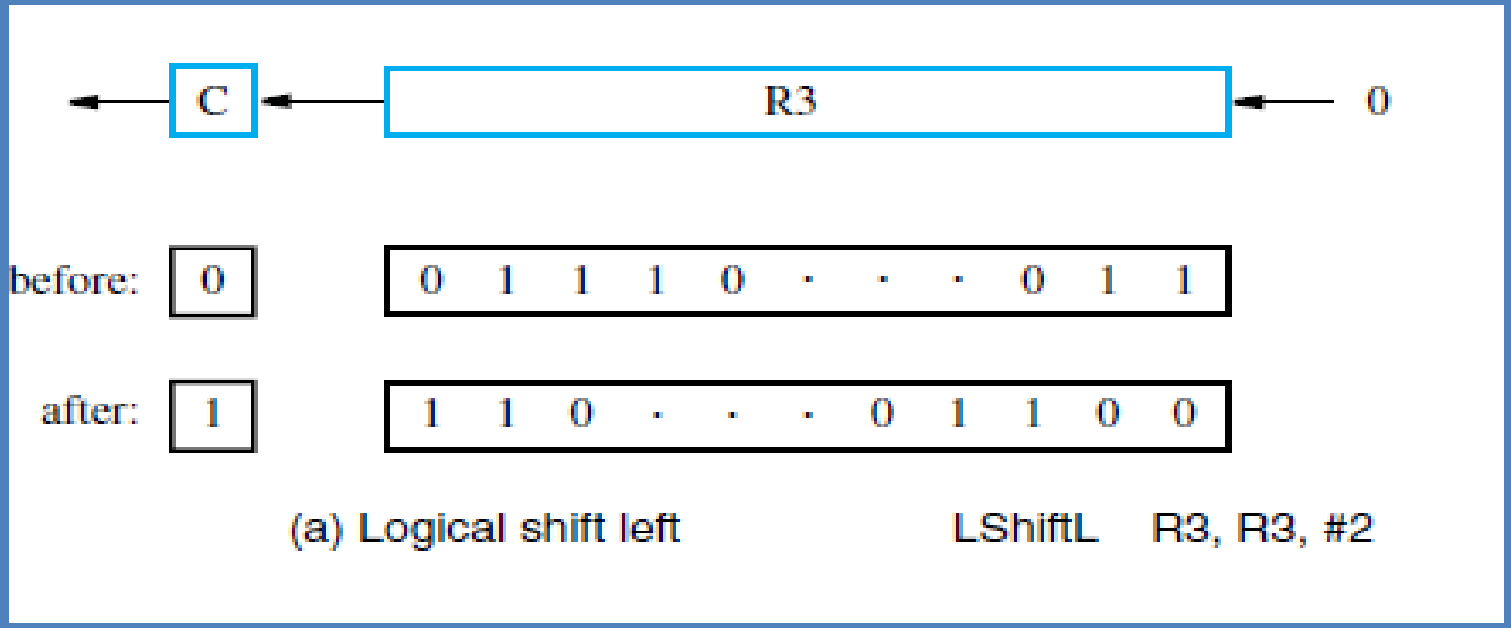
- **Instructions to shift the content of an operand Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in a end of the word determines the type of shift used.**
- **Shift instruction may specify either Logical shifts, Arithmetic shifts ,or rotate type operations.**

Instructions are as follows:

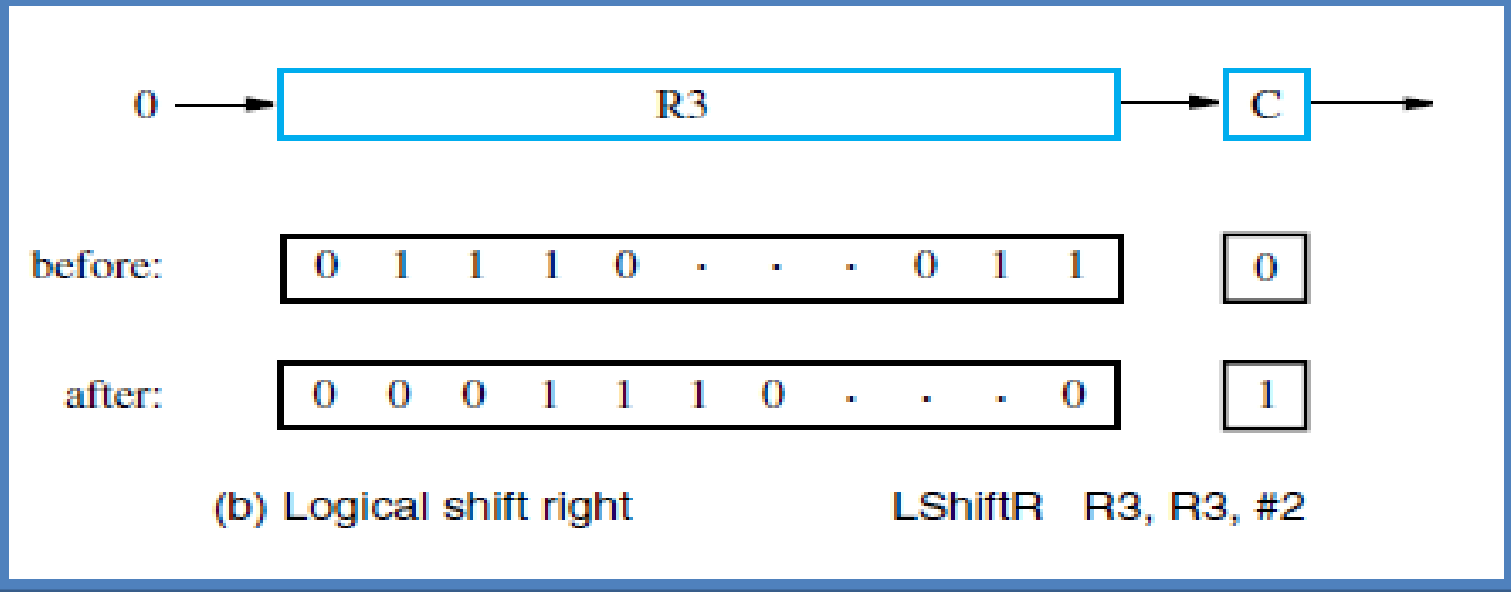
1. **Logical shift Right (SHR)**
2. **Logical shift Left (SHL)**

Logical shift inserts 0 to the end bit position. The end position is the leftmost bit for shift right and the right most bit position for the shift left.

Data Manipulation Instructions



(a) Logical shift left LShiftL R3, R3, #2



(b) Logical shift right LShiftR R3, R3, #2

Data Manipulation Instructions

3. Arithmetic Shift Right (SHRA)

4. Arithmetic Shift Left (SHLA)

Arithmetic shift right instruction must preserve the sign bit in leftmost position. The sign is shifted to the right together with the rest of the number, but the sign bit itself remain unchanged.

5. Rotate Right(ROR)

6. Rotate Left (ROL)

Rotate instructions produces a circular shift. Bits shifted out at one end of the word are not lost as in a logical shift but are circulated back into the other end.

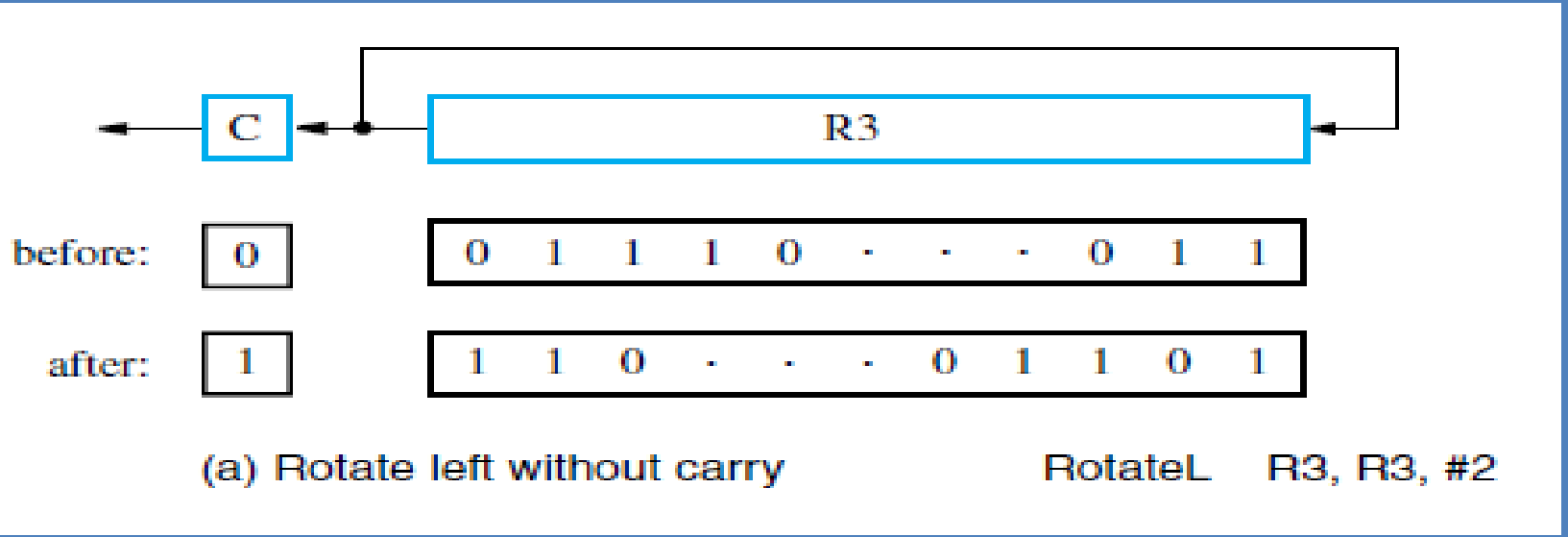
Data Manipulation Instructions

7. Rotate right through carry (RORC)

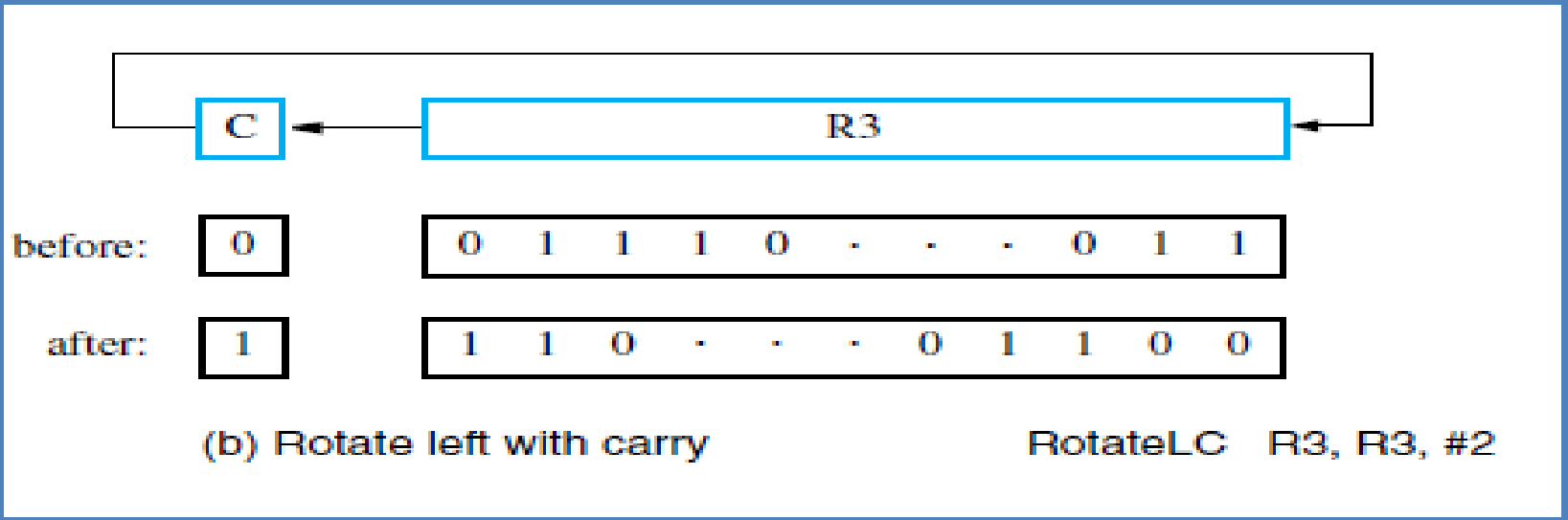
8. Rotate left through carry (ROLC)

The rotate through carry instruction transfers the carry bit into the rightmost bit position of the register, transfers the leftmost bit position into the carry , and at the same time, shifts the entire register to the left.

Data Manipulation Instructions

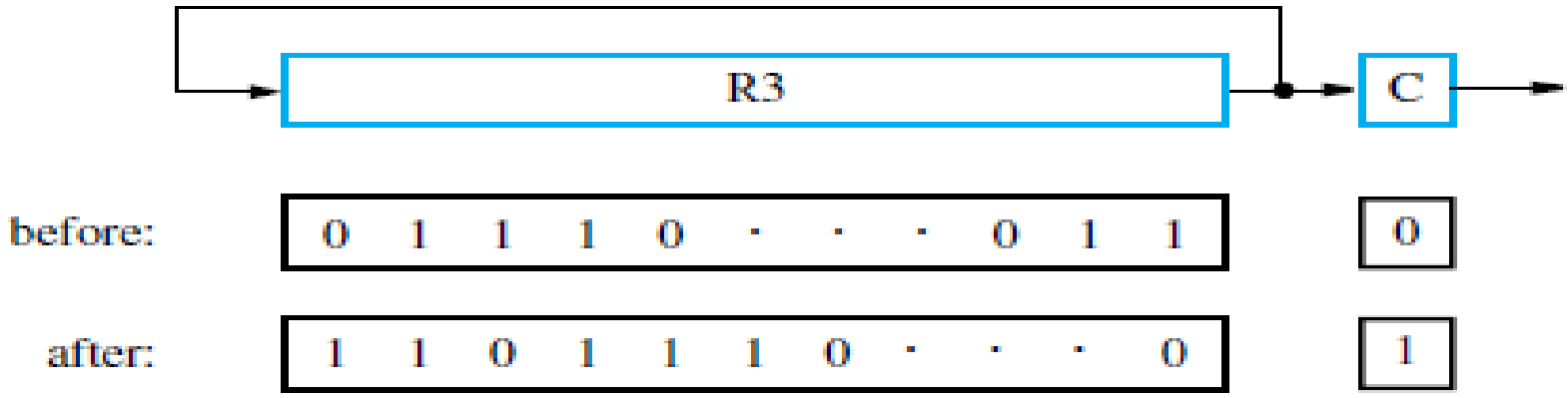


(a) Rotate left without carry RotateL R3, R3, #2

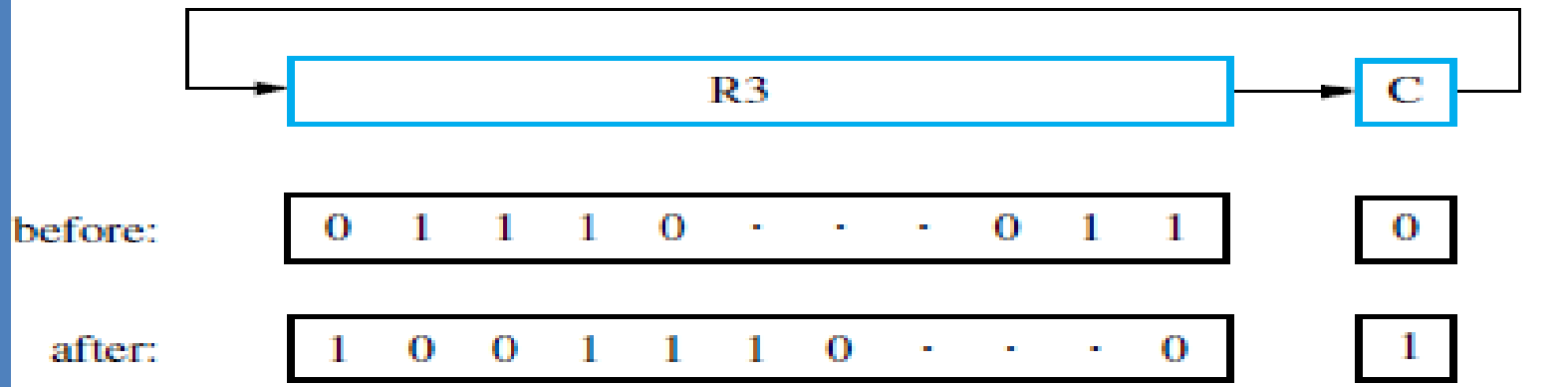


(b) Rotate left with carry RotateLC R3, R3, #2

Data Manipulation Instructions



(c) Rotate right without carry RotateR R3, R3, #2



(d) Rotate right with carry RotateRC R3, R3, #2

Data Manipulation Instructions

Some computer have a multiple-field format for the shift instructions

OP REG TYPE RL COUNT

OP is the **operation code** field,

REG is a **register address** that specify the location of the operand,

TYPE is a 2-bit field specifying the four different types of shifts,

RL is a 1-bit field specifying a shift right or left,

COUNT is a no. of shifts.

Program Control

Program control instructions specify conditions for altering the content of the program counter , while data transfer and manipulation instructions specify conditions for data-processing operations.

NAME	MNEMONICS
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by Subtraction)	CMP
Test (By ANDing)	TST

Program Control

Branch and Jump instructions may be conditional and unconditional.

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
	Unsigned compare conditions $(A - B)$	
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
	Signed compare conditions $(A - B)$	
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Program Control

Status bit Conditions

- Status bits are also called condition code bit or flag bit.
- The four status bits are symbolized by C, S, Z and V.
- The bits are set or cleared as a result of an operation performed in the ALU

Program Control

- Bit C (carry) : set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
- Bit S (sign) : set to 1 if F7 is 1. It is set to 0 if the bit is 0.
- Bit Z (zero) : set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise.
- Bit V (overflow) : set to 1 if the exclusive-OR of the last two carries (C8 and C7) is equal to 1, and cleared to 0 otherwise.

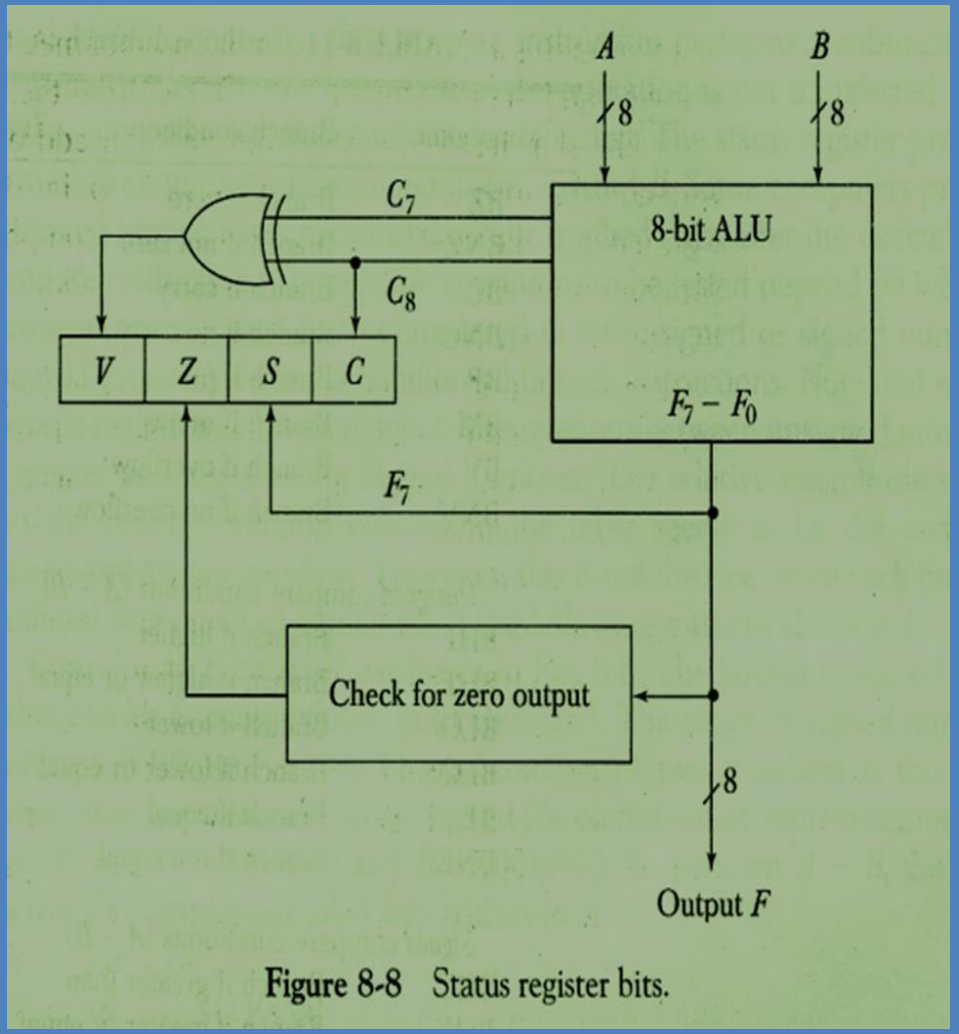


Figure 8-8 Status register bits.

Subroutine Call and Return

Subroutine: It is a self-contained sequence of instructions that performs a given computational task.

During the execution of a program, a subroutine may be called to perform its function many times at various points in the main program.

Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions.

After the subroutine has been executed, a branch is made back to the main program.

Subroutine Call and Return

A subroutine call is implemented with the following micro-operations:

CALL:

SP ← SP-1 Decrement stack point

M[SP] ← PC Push content of PC onto the stack

PC ← Effective Address : Transfer control to the subroutine

RETURN:

PC ← M[SP] : Pop stack and transfer to PC

SP ← SP+1 : Increment stack pointer

Program Interrupt

- Transfer program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed
- The interrupt procedure is , in principle, quite similar to a subroutine call except for three variations
    1. The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction (Except for software interrupt).
    2. The address if the interrupt service program is determined by the hardware rather than from the address field of an instruction.

Program Interrupt

3. An interrupt procedure usually store all the information necessary to define the state of the CPU rather than storing only the program counter.

Types of Interrupts

Generally there are three types of Interrupts those are Occurred For Example

1) Internal Interrupt
2) External Interrupt.
3) Software Interrupt.

Types of Interrupts

1. Internal Interrupt (traps)
   - Internal interrupts arise from illegal or erroneous use of an instruction or data.
   - Example: divide by zero, stack overflow.
2. External Interrupt
   - External interrupts come from I/O devices, from a timing device, from circuit monitoring the power supply, or from any other external source.
   - Example: I/O devices requesting for transfer of data, power failure.

Types of Interrupts

3. **Software Interrupt.**
   - A software interrupt is initiated by executing an instruction.
   - They can be used by a programmer to cause interrupts if need be.

     The primary purpose of such interrupts is to switch from user mode to supervisor mode.

Types of Interrupts

3. Software Interrupt.

Example:

```
#include <dos.h>
        union REGS regs;
        main(){
                regs.h.ah = 0;
                regs.h.al = 1;
                int86( 0x10, &regs, &regs );
                printf("Fourty by Twenty-Five color mode.");
        }
```

CISC (Complex Instruction set for Computer)

- CISC is a Complex Instruction Set Computer. It is a computer that can address a large number of instructions.
- Because of cheaper digital hardware, earlier it was trend to provide more and more complex operation directly in the form of processor instruction.
- The translation from high level to machine level language program is done by means of a compiler , as most high level statements can be translated to a few instructions.

CISC (Complex Instruction set for Computer)

- CISC architecture support variable length instruction format.
- For Example, Instructions that require register operands may be only two byte in length, but instruction that need two memory addresses may need five bytes to include entire instruction code.
- If computer has 32-bit word(four byte), the first instruction occupies half a word, while the second instruction needs one word in addition to one byte in the next word.

CISC (Complex Instruction set for Computer)

- CISC processor provide direct manipulation of operand residing in memory.
- For example, ADD instruction may specify one operand in memory through index addressing and second operand in memory through a direct addressing.

CISC (Complex Instruction set for Computer)

Then major characteristics of CISC architecture are:

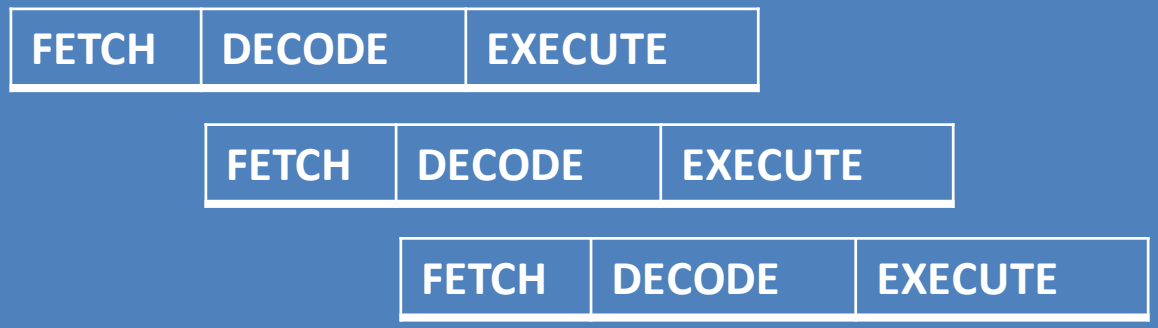
- A large number of instruction set.
- Variable length instruction.
- Specialized instructions which are used infrequently.
- Large variety of addressing modes.
- Instructions that manipulates operand directly in memory.

RISC (Reduced Instruction set for Computer)

- RISC processor supports only commonly used instructions which reduces the complexity of control unit of processor and increase execution speed.
- Small set of instructions of a typical RISC processor consists mostly of register-to-register operations, with only load and store operations for memory access.
- Only few addressing modes like register, immediate and relative are supported.
- The Instruction format are simple, which makes decoding faster and most instructions are fixed length.

RISC (Reduced Instruction set for Computer)

- The RISC processor operate much faster then CISC processor. They use massive pipelining in their architecture to improve performance.



RISC (Reduced Instruction set for Computer)

Then major characteristics of RISC architecture are:

- Relatively small instruction set.
- Few addressing modes.
- Fixed length , easily decoded instruction format.
- Single cycle instruction execution.
- Memory access limited to load and store instructions.
- A relatively large number of registers in the processor unit.
- All operations done within the registers of the CPU.
- Use of overlapped register windows to speed-up procedure call and return.